VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

NOVÉ TECHNIKY EVOLÚCIE CELULÁRNYCH AUTOMATOV

NEW EVOLUTIONARY ALGORITHMS FOR DESIGNING CELLULAR AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM ORMANDY**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Táto práca sa zaoberá evolučným návrhom prechodových funkcií celulárnych automatov postavených na podmienkových pravidlách. Práca predstavuje nový algoritmus ESP a jeho porovnanie s existujúci evolučnými technikami, konkrétne evolučnou stratégiou a genetickým algoritmom. Ako prípadové štúdie riešené v navrhovaných celulárnych automatoch, boli zvolené replikujúce sa štruktúry, pohybujúce sa objekty a vývoj vzorov.

Abstract

This thesis describes an evolutionary design of state-transition functions in cellular automata built on conditionally matching rules. It presents a new algorithm ESP and its comparison with already existing evolutionary techniques, specifically the evolutionary strategy and genetic algorithm. Chosen Case studies include self-replicating structures, moving objects and development of patterns.

Kľúčové slová

celulárny automat, podmienkové pravidlo, genetický algoritmus, evolučná stratégia

Keywords

cellular automaton, conditionally matching rule, genetic algorithm, evolutionary strategy

Citácia

ORMANDY, Adam. *Nové techniky evolúcie celulárnych automatov*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bidlo Michal.

Nové techniky evolúcie celulárnych automatov

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Michala Bidla Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Adam Ormandy
16. mája 2017

Poďakovanie

Ďakujem svojmu vedúcemu za pomocnú ruku, a za to že mal so mnou trpezlivosť. Táto práca bola podporená Ministerstvem školství, mládeže a tělovýchovy České republiky v rámci projektu veľkých infraštruktúr pre výskum, vývoj a inovácie „IT4Innovations národní superpočítačové centrum“ – LM2015070.

Obsah

Kapitola 1

Úvod

V súčastnosti existujú rôzne platformy pre realizáciu univerzálnych či aplikačne špecifických výpočtov. Z najčastejšie používaných môžme spomenúť napr. univerzálne procesory alebo technológiu FPGA. Tieto platformy sa vyznačujú dobre zavedenými metodikami realizácie rôznych algoritmov, v niektorých prípadoch je dokonca možné vytvoriť výslednú implementáciu cieľového riešenia automatizovane (napr. v prípade FPGA). Tieto platformy je možné v dnešnej dobe považovať za konvenčné technológie. Existujú však aj ďalšie možnosti, kde pre realizáciu výpočtu použijeme rôzne experimentálne platformy. Medzi ne patria napr. celulárne automaty [13], hardwarova architektúra Cell Matrix [6] alebo programovateľné tranzistorové polia (FPTA) [16].

Táto práca sa venuje výhradne celulárnym automatom a zameriava sa na automatizáciu ich návrhu pre vybrané prípadové štúdie. Problémom celulárnych automatov je to, že k ich programovaniu neexistuje efektívny univerzálny postup, keďže výpočet pomocou tejto platforme je do značnej miery emergentná vlastnosť celulárnych automatu. V minulosti sa ako riešenie problému programovanie celulárnych automatov osvedčilo použitie evolučných algoritmov a preto je táto práca zameraná práve na ne.

Cieľom tejto práce je naštudovať problematiku evolučného návrhu celulárnych automatov, na základe existujúcich techník navrhnúť evolučné algoritmy, implementovať ich a tieto implementácie porovnať na vybraných problémoch.

Štruktúra tejto práce je nasledujúca: V kapitole 2 sa venujem princípom a histórií celulárnych automatov a podmienkových pravidiel. V kapitole 3 sú vysvetlené princípy použitých evolučných algoritmov. Kapitola 4 obsahuje popis jednotlivých experimentov. V kapitole 5 sú uvedené výsledky jednotlivých experimentov a diskusia o týchto výsledkoch. Posledná kapitola 6 obsahuje záver.

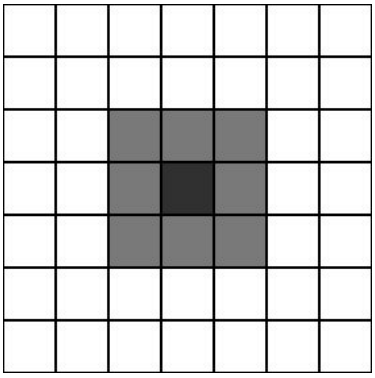
Kapitola 2

Celulárne automaty

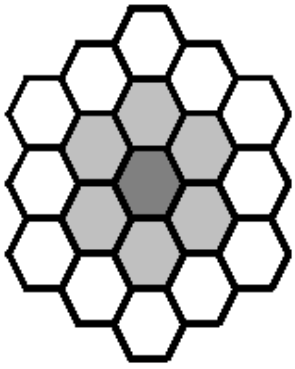
2.1 Princíp

Celulárne automaty predstavujú ideálny matematický model reálnych systémov. V tomto modeli sú priestor, čas a dokonca aj stav jednotlivých častí diskrétnymi veličinami. CA sa skladajú z dvoch základných komponentov, a to z mriežky buniek a prechodovej funkcie [20].

Mriežka buniek nie je obmedzená v počte dimenzií, ale väčšina CA používa 1 a 2-dimenzionálne mriežky. Tvar buniek taktiež nie je nijak obmedzený, ale v 2D automatoch sú poväčšine použité bunky z tvarom štvorca. Ak je už tvar a počet dimenzií akýkoľvek, všetky CA majú jednu spoločnú vlastnosť, a to že bunky majú vždy práve 1 z možných stavov.



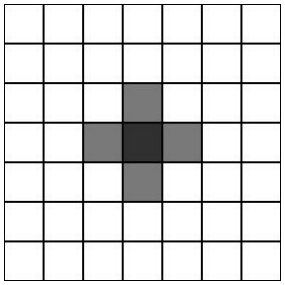
Obr. 2.1: Štvorcové CA.



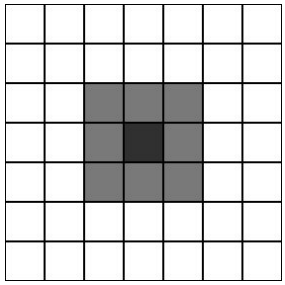
Obr. 2.2: Hexagonálne CA [19].

Stav buniek CA je ovplyvnený dvoma faktormi, prechodovou funkciou a bunkovým okolím. Rozsah a tvar bunkového môže byť rôzny, ale 2 najpoužívanejšie okolia sú Von Neumanovo a Moorove. Von Neumanovo okolie, inak zvané aj 5-okolie, je pomenované po Johnovi von Neumannovi, otca myšlienky CA a toto okolie bolo použité v prvom CA [1]. Moorovo okolie, inak zvané aj 9-okolie, je použité v „Game of Life“ [7] [11] a ďalších z nej vychádzajúcich CA. Grafické príklady okolí sú zobrazené na obr. 2.3, 2.4 a 2.5.

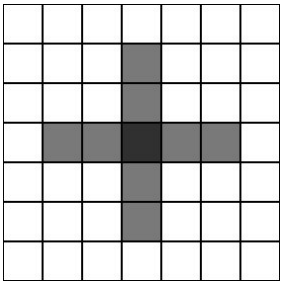
Druhým faktorom ovplyvňujúcim stav buniek a nutnou súčasťou každého CA je prechodová funkcia. Prechodové funkcie môžu byť reprezentované mnohými spôsobmi, napr. rule-stringom [7] [11], podmienkovými pravidlami [4] alebo najčastejšie používaný spôsobom, tabuľkou. Tabuľka zvyčajne obsahuje 2 stĺpce, v prvom sa nachádzajú bunkové okolia



Obr. 2.3: Von Neumanovo okolie.



Obr. 2.4: Moorove okolie.



Obr. 2.5: Rozšírené Von Neumanovo okolie.

a v druhom stĺpci budúci stav. Prechodová funkcia funguje tak, že funkcia dostane na vstup súčasné bunkové okolie, ktoré vyhľadá v tabuľke, a bunke pridelí pre nasledujúcu generáciu stav zodpovedajúci danému riadku tabuľky.

Dôležitou vlastnosťou CA je synchrónnosť a uniformnosť. Synchrónny CA je taký, ktorého bunky prechádzajú do nasledujúceho kroku (menia svoj stav) naraz, to dovoľuje použitie paralelizmu. Uniformný CA je zasa taký, v ktorom každá bunka používa tú istú prechodovú funkciu. Najpoužívanejšie CA, ako napríklad „Game of Life“ a CA použitý v tejto práci, sú synchrónne a uniformné.

2.2 História

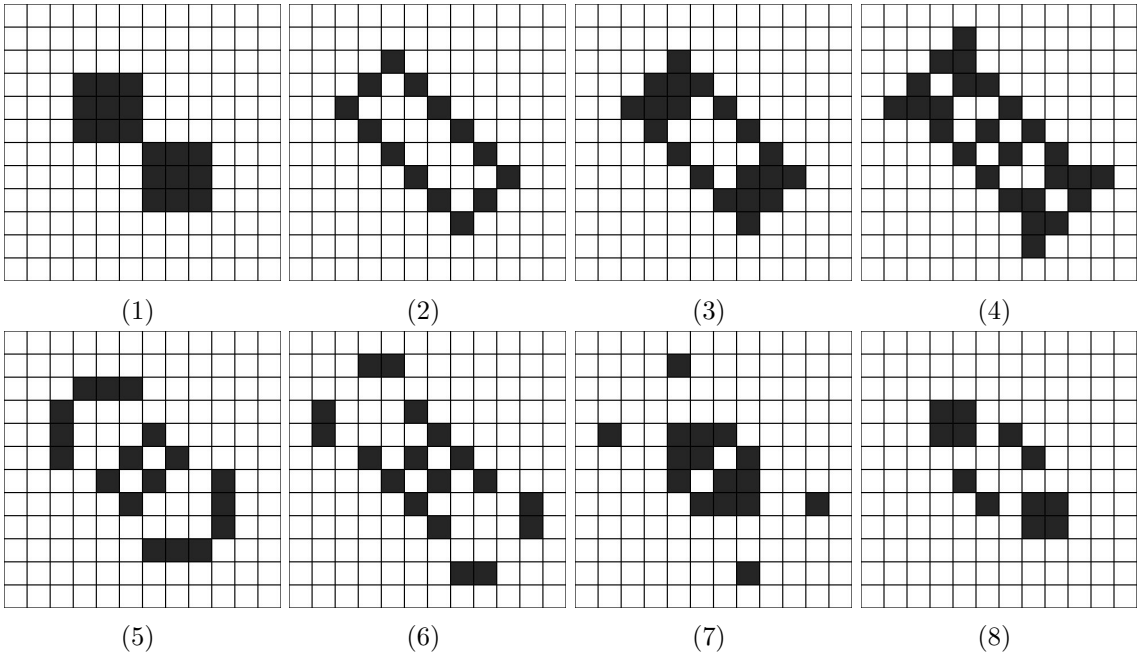
Myšlienka celulárnych automatov vznikla v 40. rokoch v Národnom Laboratóriu Los Alamos. Otcom tejto myšlienky je matematik John von Neumann, ktorý pracoval na probléme seba-replikujúcich sa systémov. Najprv pracoval so systémom, kde jeden robot postaví svoju kópiu, tkz. kinematický model [12]. Toto sa ukázalo ako moc zložité. Vtedy mu kolega, matematik Stanisław Ulam poradil použiť seba-replikujúci automat postavený na mriežkovej štruktúre z buniek [15].

Tento prvý celulárny automat bol univerzálny Turingov stroj. Používal 2-dimenzionálnu bunkovú mriežku, mal až 29 stavov, ako bunkové okolie Von Neumann použil 5-okolie [13] a obsahoval okolo 200 000 buniek [15]. Tento CA však nebol nikdy reálne zostrojený, lebo vtedajšia technika nezvládala objem výpočtov, nutný k jeho chodu [15].

Ďalším skokom vo vývoji CA bolo objavenie pravidiel pre Conwayovu Hru Života (Conway's Game of Life). Matematik John H. Conway sa zaoberal myšlienkou CA a pri pokusoch zjednodušiť pôvodný Von Neumannov automat objavil pravidlá [7] tejto hry.

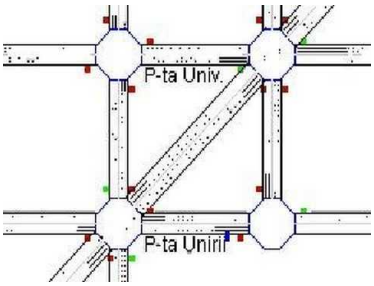
Hra Life používa synchrónny uniformný automat z 9-okolím. Je to hra 0 hráčov, keďže hráč z hrou interaguje len nastavením počiatočného stavu CA. V priebehu hry je stav buniek určovaný nasledujúcimi pravidlami:

- Ak je počet živých buniek v okolí živej bunky menší ako 2, bunka umiera.
- Ak sú v okolí živej bunky 2 alebo 3 živé bunky, bunka prežíva do ďalšej generácie.
- Ak sú v okolí živej bunky viac ako 3 živé bunky, bunka umiera.
- Ak sú v okolí mŕtvej bunky 3 živé, bunka oživne v nasledujúcej generácií.



Obr. 2.6: Oscilátor v Game of Life.

Celulárne automaty našli za svoju dlhoročnú históriu mnoho praktických využití, napr. kryptografií ako generátor náhodných čísel [17], na detekciu a opravu chýb [5]. Ďalším použitím sú simulácie dopravy, epidémií [14] a fyzikálnych javov, keďže celulárne automaty dobre simulujú reálny svet.



Obr. 2.7: Simulácia dopravy pomocou CA.

2.3 Podmienkové pravidlá

Podmienkové pravidlá predstavujú jednu z techník reprezentácie prechodových funkcií CA, ktorá bola prvý krát predstavená v [3]. Nasledujúce fakty vychádzajú z uvedeného článku a popisujú podmienkové pravidlá tak, ako budú aplikované v tejto práci.

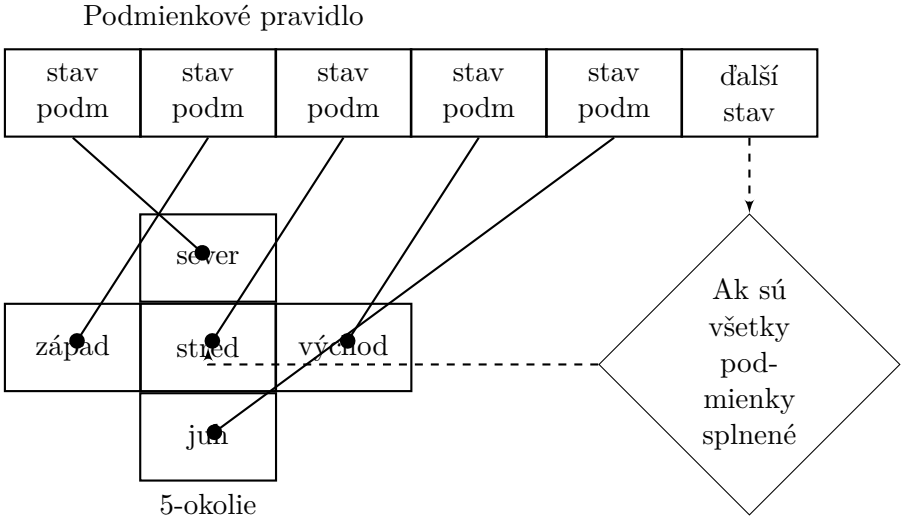
Tradičné reprezentovanie prechodovej funkcie CA tabuľkou má jeden problém. Pri náraste počtu možných stavov rastie počet kombinácií exponenciálne. To robí design tejto prechodovej funkcie veľmi náročný. Možným riešením by bolo špecifikovať iba istú podmnožinu pravidiel, napr. iba také pravidlá kde dochádza k zmene stavu, ale problémom je, ako nájsť takéto pravidlá pri komplexných CA.

Riešením tohto problému môže byť práve reprezentovanie prechodovej funkcie pomocou podmienkových pravidiel (PP). PP sú veľmi podobné tradičnej tabuľkovej reprezentácií prechodovej funkcie. Rozdiel je v tom, že jedno PP môže vyjadrovať niekoľko tabuľkových pravidiel.

Každé PP pravidlo sa skladá z 2 častí, podmienkovej a ďalšieho stavu. Podmienková časť obsahuje dvojice stavov a podmienok pre každú bunku z bunkového okolie. Podmienky môžu byť: rovná sa (==), nerovná sa (!=), viac alebo rovno (>=) a menej alebo rovno (<=). Ďalší stav je bunke pridelený, ak je splnená každá podmienka z podmienkovej časti.

Podmienky sú kódované nasledujúcim spôsobom:

- 0 je stav bunky >= podmienka
- 1 je stav bunky <= podmienka
- 2 je stav bunky == podmienka
- 3 je stav bunky != podmienka

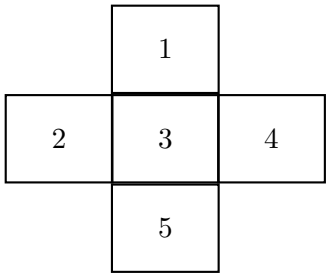


Obr. 2.8: Namapovanie PP pravidla na Von Neumanovo okolie.

Prechodová funkcia je tvorená konečných počtom PP pravidiel. K tomu aby sme zistili nasledujúci stav bunky je potrebné použiť nasledujúci postup. Berieme postupne jednotlivé PP pravidlá. Jednotlivé podmienky PP pravidla sú vyhodnocované vzhľadom na súčasné bunkové okolie. Ak sú všetky podmienky splnené, budúci stav bunky bude určený podľa tohto PP pravidla a proces hľadania končí. Ak žiadne z PP pravidiel vo funkcií nesedí na dané bunkové okolie, stav bunky sa nezmení.

Prechodová funkcia:

#1	<= 3	!= 3	!= 3	>= 4	== 0	4
#2	>= 0	== 2	<= 5	!= 4	>= 1	2
#3	!= 4	== 0	>= 3	>= 2	== 1	3



5-okolie

Obr. 2.9: Príklad prechodovej funkcie s 3 PP pravidlami a okolím.

Ako príklad vyhodnocovania prechodovej funkcie postavenej na PP môžme použiť funkciu a okolie z obrázka 2.9. Vyhodnocovanie začneme u prvého PP pravidla. Prvá testovaná podmienka má tvar 1 <= 3, takže podmienka je platná. Druhá podmienka má tvar 2 != 2, podmienka je nesplnená a vyhodnocovanie tohto pravidla končí. Začne vyhodnocovanie druhého pravidla, podmienky sú 1 >= 0, 2 == 2, 3 <= 5, 1 != 4 a 5 >= 1. Všetky tieto podmienky platia, preto bude mať bunka v nasledujúcom kroku stav 2. Ak by ani jedno PP pravidlo nebolo platné, tak by si bunka zachovala pôvodný stav.

Prechodová funkcia postavená na PP pravidlá majú niekoľko výhod. Je výrazne menšia, lebo jedno PP pravidlo reprezentuje niekoľko tabuľkových pravidiel. Ďalej je deterministická, lebo poradie vyhodnocovania PP pravidiel je presne dané. A v prípade potreby sa dajú PP pravidlá jednoducho prekonvertovať na tabuľkové, a to tak, že vygenerujeme všetky tabuľkové pravidlá, ktoré sú reprezentované jednotlivými PP pravidlami.

Kapitola 3

Evolučné algoritmy

3.1 Princíp

Na poli umelej inteligencie sú evolučné algoritmy (EA) druhom stochastických optimalizačných algoritmov inšpirovaných biologickou evolúciou. EA reprezentujú potencionálne riešenia pomocou jednoduchých štruktúr podobným chromozómom [18]. Tieto chromozómy sú potom postupne upravované tak aby vyjadrovali lepšie a lepšie riešenia.

Všeobecný princíp EA je nasledovný:

1. Náhodne inicializuj prvotnú generáciu jedincov (chromozómov).
2. Ohodnoť jedincov na základe ich fitness, kde fitness je miera funkčnosti daného riešenie.
3. Skontroluj či nenastala ukončujúca podmienka.
   - Ak áno, ukonči evolúciu.
   - Ak nie, pokračuj.
4. Vyber rodičov nasledujúcej generácie na základe ich fitness.
5. Vytvor nasledujúcu generáciu upravením rodičov.
6. Vráť sa na krok 2.

Ukončovacou podmienkou môže byť mnoho vecí, napr. nájdenie dostatočne funkčného riešenia, vyčerpanie zdrojov, dosiahnutie maximálneho počtu generácií alebo ak sa nájdené riešenia nezlepšujú.

Je dôležité povedať že nie vždy je začiatočná populácia náhodná. V prípade keď máme informáciu, kde sa v priestore riešení nachádzajú platné riešenia, tak tomu môžme prispôsobiť začiatočnú populáciu. Informácia o tom kde sa nachádzajú riešenie je však málokedy k dispozícií, preto sa väčšinou používa náhodná začiatočná populácia.

Na ohodnotenie jedincov je použitá fitness funkcia, ktorá hodnotí jedincov na základe vhodnosti riešenia, ktoré daný jedinci reprezentujú. Čím väčšia fitness, tým lepšie riešenie riešenie jedinec prestavuje a tým má jedinec väčšiu šancu byť rodičom nasledujúcej generácie.

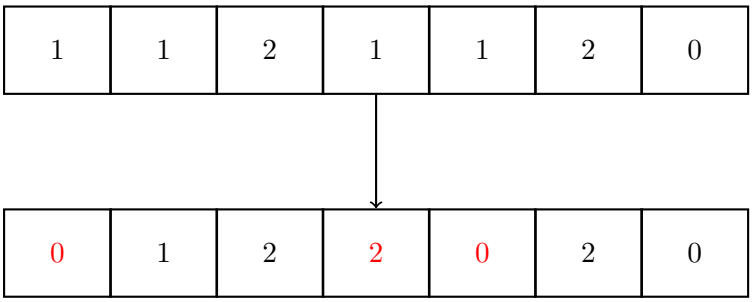
Dobrá fitness funkcia má 2 vlastnosti. Prvou je rýchlosť. Tú je potreba lebo pre nájdenie riešenia pomocou EA je často nutných mnoho generácií. Druhou je dobré škálovanie riešení,

ktoré je nutné k tomu, aby sa riešenia približovali k správnemu riešeniu. Preto je dôležité, aby problém pre ktorý je hľadané riešenie, nemal len dobré a zlé riešenia, ale rôzne dobré riešenia.

Problém EA je tendencia uviaznuť v lokálnom maxime, kde uviaznutie znamená stav, kedy evolúcia uviazne v takom riešení, ktoré má lokálne vysokú fitness ale nie je riešeným problému. Tento problém je spojený z tvarom fitness priestoru nad priestorom riešení a tendenciu uprednostnia krátkodobých ziskov fitness nad dlhodobými.

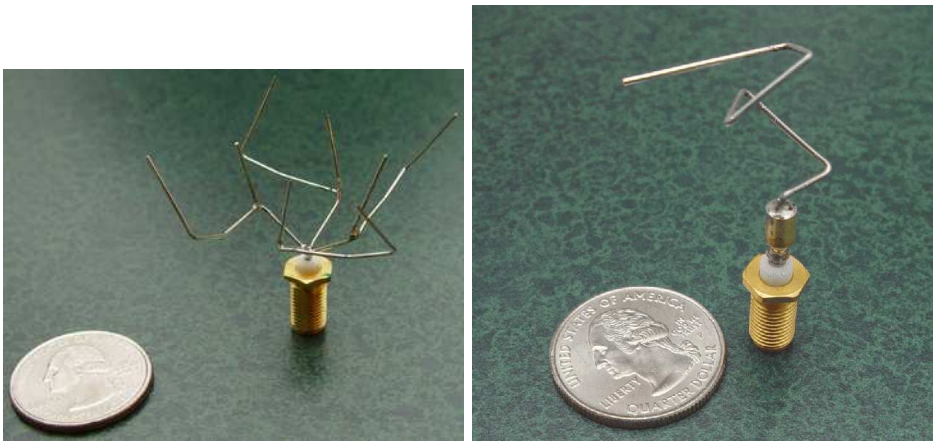
Vytvorenie novej generácie simuluje 2 základné kamene evolúcie, prirodzený výber a mutácie. Prirodzený výber je simulovaný tak, že jedinci majú rôznu šancu stať sa rodičmi pre nasledujúci generáciu. To ktorý jedinci sa stanú rodičmi je dané použitým EA.

Po výbere rodičov je nutné vytvoriť novú generáciu jedincov. Tá je zvyčajne vytvorená zmutovaním rodičov, kde mutácie majú formu zmeny častí chromozómu jedinca. V závislosti na použitej reprezentácií chromozómu, môžu byť použité aj ďalšie metódy, ako inverzia chromozómu, crossing-over alebo metódy podobné pohlavnému rozmnožovaniu. Dôležité je aj to, či je možný elitizmus, teda či najlepší chromozóm sa bez zmeny dostane do ďalšej generácie.



Obr. 3.1: Mutácia

EA majú mnoho použití, napr. design vesmírnych antén [9], bio-informatike, návrhu rozvrhu komunikácie s vesmírnymi sondami [8], návrhu hardware a mnohých ďalších.



(a) (b)

Obr. 3.2: Antény navrhnuté pomocou EA [9].

3.2 Evolučná stratégia

Evolučná stratégia (ES) patrí medzi historicky prvé úspešné stochastické algoritmy a v súčastnosti patrí medzi dobre rozvinuté stochastické metódy. ES vychádza zo všeobecných predstáv prirodzeného výberu, avšak omnoho vágnejších ako napríklad pri genetickom algoritme [10]. Preto sa ES zaraďuje medzi nejednoduchšie EA.

ES bola navrhnutá v 60. rokoch pánmi Ingo Rechenbergom a Hans-Paul Schwefelom, keď sa zaoberali experimentmi v aerodynamickom tunely, pri ktorých sa snažili optimalizovať tvar telies tak, aby mali čo najmenší odpor vo vzdušnom prúdení. Pretože intuitívny postup ani gradientové metódy neboli veľmi úspešné, začali sa zaoberať myšlienkou náhodných zmien parametrov, podobne ako je to pri mutáciách, v priebehu optimalizačného procesu.

t:=0;
P:={náhodne generovaná populácia chromozómov};
while t<t_{max} **or** dostatočne dobré riešenie nie je nájdené **do**
  t:=t+1;
  ohodnoť každý chromozóm z P funkčnou hodnotou (fitness);
  Q:={vyber najlepšieho jedinca z populácie P};
  P:={mutovaním rodičovského chromozómu Q vytvor novú populáciu};
end while

Algoritmus 1: Pseudokód ES použitej v tejto práci.

Ako je vidieť z popisu algoritmu 1, ES je jednoduchý EA a začína s náhodne generovanou začiatočnou populáciou. Potom sú jednotlivé chromozómy ohodnotené. Následne je z populácie vybraný najlepší chromozóm, z ktorého je pomocou mutácie vytvorená nová populácia. Tento cyklus vytvorenia, ohodnotenia a vybrania rodiča pokračuje až pokým nie je nájdené uspokojivé riešenie alebo nie je dosiahnutý maximálny počet generácií CA.

3.3 Genetický algoritmus

Genetické algoritmy [10] patria medzi najznámejšie prehľadávacie algoritmy založené na evolúcií. GA boli vyvinuté Johnom Hollandom pri pokuse vysvetliť adaptívny procesy prírodných systémov a pri návrhu im podobných umelých systémov. V súčasnej dobe patria GA k najrozšírenejším technikám využívajúcim princípov prírodného výberu a genetiky. GA a ich rozšírenia sa evolúcií podobajú viac ako väčšina iných evolučných metód. Dnes sú známe predovšetkým pre ich schopnosť riešiť širokú škálu optimalizačných problémov a robustnosť [2].

Najjednoduchšia forma GA, takzvaný *kanonický* alebo *jednoduchý* GA, je ukázaný vo výpise algoritmu 2. Jeho činnosť je nasledujúca: Gény každého jedinca v populácií sú inicializované na náhodnú hodnotu, tým sa vytvorí začiatočná populácia. Po tomto úkone je spustený hlavný cyklus GA. Na každého jedinca je aplikovaná ohodnocovacia funkcia, ktorá určí fitness daného chromozómu. Nasleduje výber rodiča alebo rodičov pre nasledujúcu generáciu. Na to je použitá turnajová selekcia, ktorá spočíva v súboji 2 alebo viacerých jedincov, kde najlepší z nich bude vybratý za rodiča. Selekcia turnajom zabezpečuje väčšiu variabilitu, lebo nielen najlepší jedinec sa môže stať rodičom. Po vybratí rodiča alebo rodi-

čov je pomocou mutácií vytvorená nová populácia. Tento cyklus sa opakuje až dokým nie je nájdené riešenie alebo kým

```
t:=0;
P(t):={náhodne generovaná populácia chromozómov};
repeat
  urči fitness jedincov P(t);
  z P(t) vyber rodiča turnajovou selekciou;
  t:= t + 1;
  repeat
    pomocou mutácie vytvor z rodiča potomka;
    ulož potomka do novej populácie P(t)
  until P(t) je naplnená potomkami
until splnená ukončujúca podmienka
```

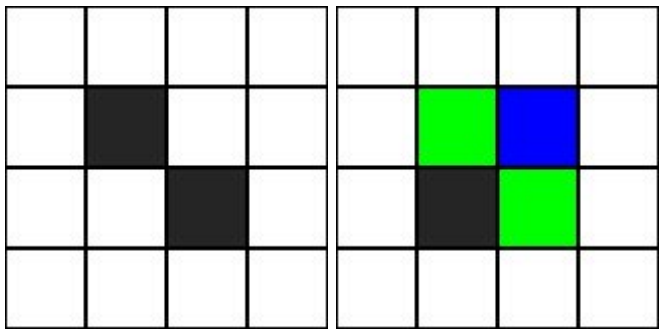
Algoritmus 2: Pseudokód algoritmu GA, ktorý je podobný algoritmu použitom v tejto práci.

Kapitola 4

Experimenty

Jednotivé EA boli použité na návrh prechodových funkcií CA. Tieto prechodové funkcie boli vyjadrené pomocou PP. CA pre ktorý boli hľadané prechodové funkcie bol ohraničený, synchrónny a uniformný. Ohraničenie bolo implementované pomocou buniek z hodnotou 0, ktoré nemohli meniť svoju hodnotu.

Fitness jednotlivých chromozómov bola určená tak, že sa počas niekoľkých generácií CA porovnával stav jednotlivých buniek CA oproti želanému výsledku, vzoru. Celková fitness chromozómu bola hodnota, kedy stav CA najviac zodpovedal želanému vzoru.



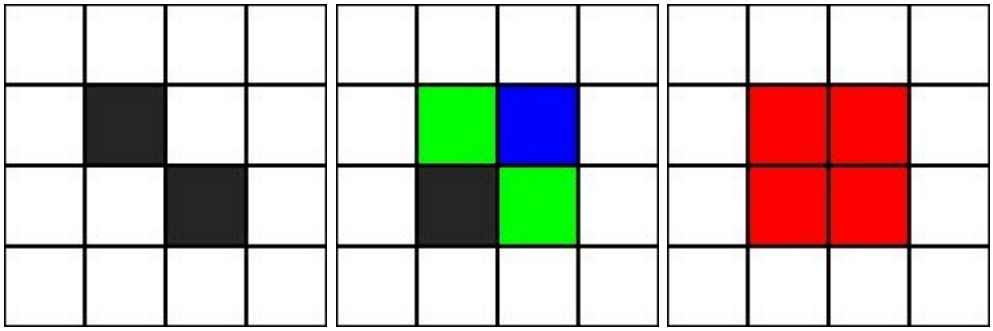
(a) Vzor (b) Stav CA

Obr. 4.1: Ukážka určovanie fitness pre jednu generáciu. Ak je obr. 4.1a náš vzor, tak CA zachytené na obr. má fitnes 14, preto lebo bunky 2 bunky nezodpovedajú vzoru.

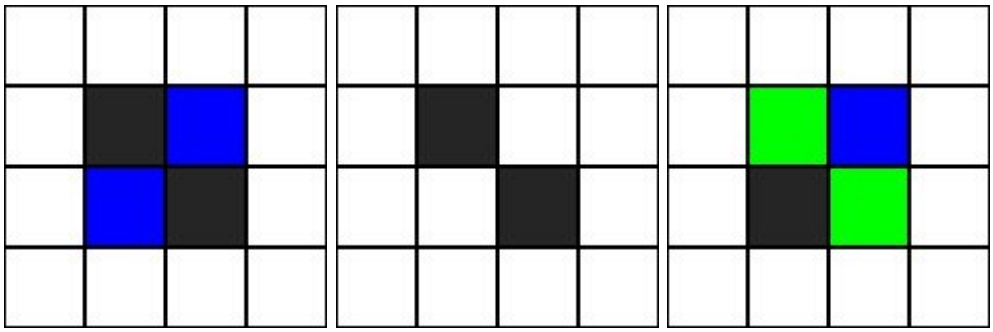
Ak EA našlo prechodové pravidlo, kde aspoň v jednej generácií stav CA plne zodpovedal želanému alebo ak bol dosiahnutý maximálny počet generácií, EA skončil.

Nájdené prechodové funkcie boli následne manuálne skontrolované, jednak ako kontrola, či CA použité v EA funguje správne, a pre nájdenie prechodových funkcií so zaujímavými vlastnosťami ako napr. stabilita vzoru, neblikavé pravidlá alebo stabilná replikácia.

Pre zníženie časovej náročnosti jednotlivých behov bol implementovaný systém nevyhodnocovania fitness pre X začiatočných generácií CA. Počet nevyhodnotených generácií bol určený samostatne pre každý experiment a vyjadroval počet generácií CA, kde nepredpokladáme, že by CA dosiahol požadovaný stav. Tento systém bol tiež použitý k zamedzeniu vývoja triviálnych riešení.



(a) počiatočný stav (b) fitness=12 (c) fitness=12



(d) fitness=14 (e) fitness=16 (f) fitness=12

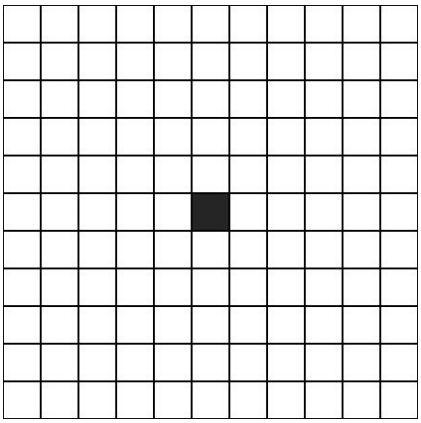
Obr. 4.2: Príklad určovania celkovej fitness chromozómu. V tomto príklade sa snažíme nájsť prechodovú funkciu pre oscilátor, ktorý sa vracia do stavu ako na obr. 4.2a a CA sa vyvíja po dobu 5 generácií. Jednotlivé generácie majú fitness 12, 12, 14, 16 a 12. 16 je najvyššia dosiahnutá fitness, preto celková fitness prechodovej funkcie bude 16.

4.1 Experiment s vývojom vzoru

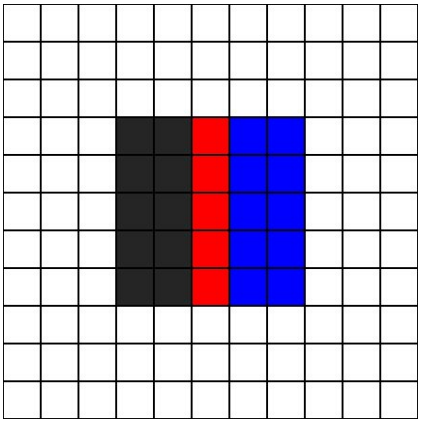
V tomto experimente mal EA za úlohu nájsť prechodovú funkciu CA, ktorá zo semiačka vytvorí vzor francúzskej vlajky. Francúzska vlajka bola vybraná z toho dôvodu, že sa používa ako benchmark na testovanie EA pracujúcich z CA.

Nastavenia EA a CA boli nasledujúce:

- veľkosť populácie: 8.
- maximálny počet generácií EA: 2 000 000.
- počet mutácií na jeden chromozóm: 1-3.
- počet PP: 30, 40 a 50.
- bunkové okolie: Von Neumannovo.
- počet krokov CA: 20.
- počet nevyhodnocovaných krokov CA: 8.
- počet stavov: 5.
- počet turnajových kôl: 2.



Obr. 4.3: Počiatočný stav CA.



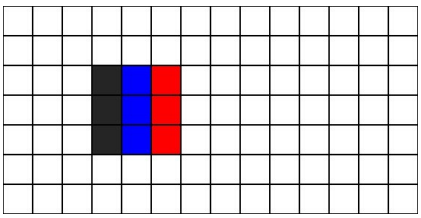
Obr. 4.4: Želaný stav CA.

Nastavenia boli zvolené na základe mojich predchádzajúcich skúseností a skúseností vedúceho tejto práce. Z nich vyplynulo napr. to že na zvýšenie počtu stavov nad nutnú mieru nezlepšuje výsledky EA a že riešenia nebývajú často nachádzané po 2 000 000 generácií. Počet krokov CA bol odhadnutý z ohľadom na výkon a pravdepodobného počtu krokov nutných na vývoj vzoru.

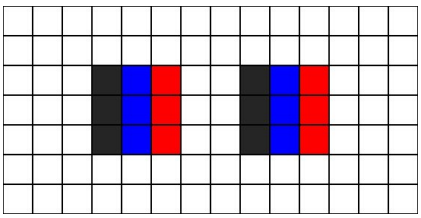
Veľmi dôležité je nastavenie počtu nevyhodnocovaných krokov CA. Počet bol vybraný jednak z výkonnostných dôvodov a preto, aby sa zamedzilo vývoju triviálnych riešení.

4.2 Experiment s replikáciou vzoru

Tento experiment bol zameraný na hľadanie takej prechodovej funkcie CA, ktorá zreplikuje počiatočný vzor vedľa pôvodného, bez toho aby bol pôvodný vzor posunutý alebo zničený.



Obr. 4.5: Počiatočný stav CA.



Obr. 4.6: Želaný stav CA.

Nastavenia EA a CA boli nasledujúce:

- veľkosť populácie: 8.
- maximálny počet generácií EA: 2 000 000.
- počet mutácií na jeden chromozóm: 1-3.
- počet PP: 30, 40 a 50.
- bunkové okolie: Von Neumannovo.
- počet krokov CA: 20.
- počet nevyhodnocovaných krokov CA: 6.

- počet stavov: 7.
- počet turnajových kôl: 2.

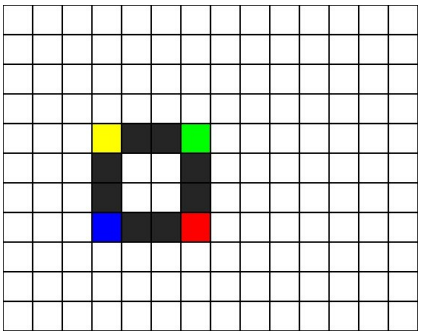
Hodnoty jednotlivých nastavení boli tak ako v predchádzajúcom experimente určené na základe skúseností a odhadu.

4.3 Experiment s gliderom

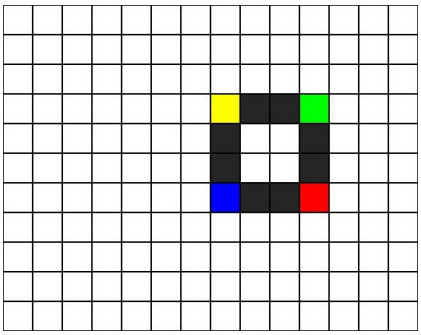
V tejto úlohe malo EA nájsť takú prechodovú funkciu CA, ktorá spraví z danej štruktúry glider. Glider, známy z hry Life, je štruktúra ktorá sa pohybuje po ploche CA. Pohyb má často formu rozloženia štruktúry a jej zloženia o kúsok ďalej.

Nastavenia EA a CA boli nasledujúce:

- veľkosť populácie: 8.
- maximálny počet generácií EA: 2 000 000.
- počet mutácií na jeden chromozóm: 1-3.
- počet PP: 30, 40 a 50.
- bunkové okolie: Von Neumannovo.
- počet krokov CA: 16.
- počet nevyhodnocovaných krokov CA: 4.
- počet stavov: 5.
- počet turnajových kôl: 2.



Obr. 4.7: Počiatočný stav CA.



Obr. 4.8: Želaný stav CA.

Kapitola 5

Experimentálne výsledky

5.1 Výsledky ES

Prvým skúmaním algoritmom bola verzia ES, ktorá za rodiča vybrala vždy prvý chromozóm z najvyššou fitness. Tento algoritmus je implementovaný vo funkcií `ES_next_gen`.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	95/96	102551.116	119549.26
40	93/96	213299.269	298281.15
50	88/96	345134.440	394598.69

Tabuľka 5.1: Výsledky experimentu s vývojom vzoru.

ES nemalo problém nachádzať riešenia pre problém vývoju vzoru, ako je vidieť na tabuľke 5.1. Zvýšenie počtu PP malo negatívny efekt na úspešnosť a aj výpočtovú náročnosť.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	26/96	789071.769	490891.74
40	32/96	865349.812	592137.01
50	36/96	891970.556	506890.64

Tabuľka 5.2: Výsledky experimentu s gliderom.

Na tabuľke 5.2 je vidieť, že ES si vedelo poradiť z nájdením prechodovej funkcie. Tabuľka zároveň ukazuje, že tento problém si vyžaduje komplexnú prechodovú funkciu, lebo zvýšenie počtu PP zvýšilo úspešnosť.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	89/96	381719.023	410894.25
40	91/96	431609.861	386651.28
50	93/96	351558.105	433979.27

Tabuľka 5.3: Výsledky experimentu s replikáciou vzoru.

Tabuľka 5.3 ukazuje, že replikácia vzoru nepredstavovala problém pre ES.

5.2 Výsledky ES_ELIT (ES s elitizmom)

Ten istý algoritmus ako v prípade ES, z tým rozdielom, že rodičovský chromozóm v nezmenenej podobe prenesený do ďalšej generácie. Implementácia je prítomná vo forme funkcie z názvom `ES_ELIT_next_gen`.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	88/96	366952.341	467358.79
40	81/96	351535.963	422520.67
50	66/96	456527.470	473835.93

Tabuľka 5.4: Výsledky experimentu s vývojom vzoru.

Tabuľka 5.4 ukazuje že ES s elitizmom bola schopná nachádzať riešenie pre problém vývoju vzoru. Zaujímavé je prepad úspešnosti o štvrtinu pri použití 50 PP, čo naznačuje že táto úloha nepotrebovala až tak komplexnú prechodovú funkciu.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	30/96	813194.333	466691.03
40	30/96	868274.567	542357.27
50	28/96	851361.035	553666.00

Tabuľka 5.5: Výsledky experimentu s gliderom.

Výsledky (tab. 5.5) ukazujú že ES s elitizmom je schopná nachádzať riešenia pre experiment z gliderom. Ukazuje sa aj to, že počet PP nemal vplyv na úspešnosť alebo výpočtovú náročnosť.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	84/96	382021.464	415944.21
40	86/96	410639.267	394944.22
50	85/96	373031.258	368952.24

Tabuľka 5.6: Výsledky experimentu s replikáciou vzoru.

ES s elitizmom (tab. 5.6) bol veľmi úspešní v nachádzaní riešení v experimente z replikáciou vzoru. Na základe smerodajných odchýlok sa ukazuje, že rozptyl v počte generácií EA nutných na nájdenie riešení bol značný.

5.3 Výsledky GA

Genetický algoritmus z turnajovou selekciou. Počet turnajových kôl bol nastavovaný pomocou makra `EA_CONTROL_SUM`. Funkcia implementujúca GA má názov `GA_next_gen`.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	4/96	1185374.658	782668.25
40	4/96	513868.214	456217.78
50	16/96	749769.562	618612.23

Tabuľka 5.7: Výsledky experimentu s vývojom vzoru

GA sa podarilo nájsť riešenia pre úlohu, ako ukazuje tabuľka 5.7 ale úspešnosť bola veľmi malá. Ďalší problém je aj to, že na nájdenie týchto riešení bolo treba použiť veľa generácií EA.

GA bol neúspešný v hľadaní riešení pre úlohu hľadania glideru. To mohlo mať niekoľko príčin, napr. nevhodnosť GA pre daný priestor riešení alebo neoptimálne nastavenie GA.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	63/96	862396.588	533008.23
40	58/96	810117.215	495627.20
50	64/96	842588.395	568456.87

Tabuľka 5.8: Výsledky experimentu s replikáciou vzoru

GA sa podarilo nájsť riešenie pre úlohu replikácie vzoru (tab. 5.8). Na tabuľke je vidieť aj to, že veľkosť chromozómu nemala vplyv na úspešnosť, výpočtovú náročnosť a ani na rozptyl.

5.4 Výsledky GA_ELIT (GA s elitizmom)

Táto implementácia GA s elitizmom je vo funkcií `GA_ELIT_next_gen`.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	80/96	322330.862	442067.53
40	81/96	457395.765	497294.34
50	65/96	528656.400	518113.89

Tabuľka 5.9: Výsledky experimentu s vývojom vzoru

Z tabuľky 5.9 je vidieť, že GA_ELIT bolo úspešné v hľadaní riešení. Pri použití chromozómov o veľkosti 50 PP nastal takmer štvrtinový prepad v úspešnosti, čo nahovára, že táto úloha nepotrebuje tak komplexnú prechodovú funkciu. Zaujímave je, že pre všetky nastavenia veľkosti chromozómu mal GA_ELIT veľký rozptyl.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	31/96	870297.290	617564.76
40	34/96	876363.705	579988.93
50	24/96	973304.083	545874.56

Tabuľka 5.10: Výsledky experimentu s gliderom

Tabuľka 5.10 ukazuje, že GA_ELIT sa podarilo nájsť riešenia pre problém. Je vidieť že to bola ťažšia úloha ako úloha vývoja (tab. 5.9) a replikácia vzoru (tab. 5.11). Je zaujímavé ako klesá odchýlka ale zvyšuje sa výpočtová náročnosť pri zvýšení počtu PP v chromozóme, čo hovorí že riešenia boli postupne bližšie pri sebe a bolo potreba viac generácií EA na ich nájdenie.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	88/96	361717.795	466764.82
40	86/96	423443.581	521204.76
50	85/96	390937.871	491083.60

Tabuľka 5.11: Výsledky experimentu s replikáciou vzoru

Ako je vidieť na tabuľke 5.11, tak algoritmus bol úspešný v hľadaní riešení a vychádzajúc z vysokej smerodajnej odchýlky, boli riešenie veľmi rozptýlené.

5.5 Výsledky ESP

Ďalšia implementácia ES, z tým rozdielom, že v tejto sa rodič vyberá tak, že rodičia sú zoradený od najlepšieho po najhoršieho a potom je vybraný náhodný chromozóm z najlepších chromozómov. Implementácia má formu funkcia z názvom `ESP_next_gen`.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	96/96	74879.500	88380.97
40	94/96	169358.798	172554.54
50	91/96	341749.363	423412.06

Tabuľka 5.12: Výsledky experimentu s vývojom vzoru

Algoritmus ESP bol veľmi úspešný v riešení úlohy vývoja vzoru. Tab. 5.12 ukazuje, že pre túto úlohu bolo potreba menšie chromozómy (30 PP), a väčšie chromozómy viedli len k výraznému zvýšeniu výpočtovej náročnosti.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	38/96	850813.605	591286.17
40	35/96	793942.085	482420.50
50	42/96	978177.762	608115.28

Tabuľka 5.13: Výsledky experimentu s gliderom

Z výsledkov (tab. 5.13) je vidieť že úloha s glidrom bola náročnejšia, ale ESP si s ňou vedel poradiť. Zaujímavé je, že pri použití 50 PP na chromozóm nastalo výrazné zvýšenie priemerného počtu generácií EA ale aj úspešnosti.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	87/96	439768.505	477316.05
40	88/96	467466.511	488132.75
50	89/96	410797.430	467890.77

Tabuľka 5.14: Výsledky experimentu s replikáciou vzoru

V experimente z replikáciou vzoru (tab. 5.14) bola dosiahnuté ESP algoritmom veľmi dobré výsledky. Zaujímavé je, že veľkosť chromozómu nemala veľký efekt na úspešnosť.

5.6 Výsledky ESP_ELIT (ESP s elitizmom)

Funkcia z týmto algoritmom má názov `ESP_ELIT_next_gen`.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	84/96	313700.250	409756.26
40	78/96	274926.615	381098.69
50	73/96	478121.397	461001.36

Tabuľka 5.15: Výsledky experimentu s vývojom vzoru

Z tabuľky 5.15 vyplýva, že optimálny počet PP pri použití ESP_ELIT bol okolo 30. Dosiahli sme relatívne vysokú úspešnosť a priemerná výpočtová zložitosť nepresahuje štvrtinu maximálneho počtu generácií EA. Smerodajná odchýlka naznačuje že výsledky neboli blízko seba.

počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	22/96	736286.954	578116.54
40	32/96	813215.968	607313.66
50	30/96	853608.100	669209.12

Tabuľka 5.16: Výsledky experimentu s gliderom

Z tabuľky 5.16 môžme vyčítať to, že toto bola ťažšia úloha oproti napr. úlohe vývoja vzoru (tab. 5.15) lebo sme dosiahli menšiu úspešnosť, priemerný čas na nájdenie bol vyšší a väčšia veľkosť chromozómu mala na výsledky pozitívny dopad. Rozptyl v počte generácie bol veľký, ako naznačuje smerodajná odchýlka.

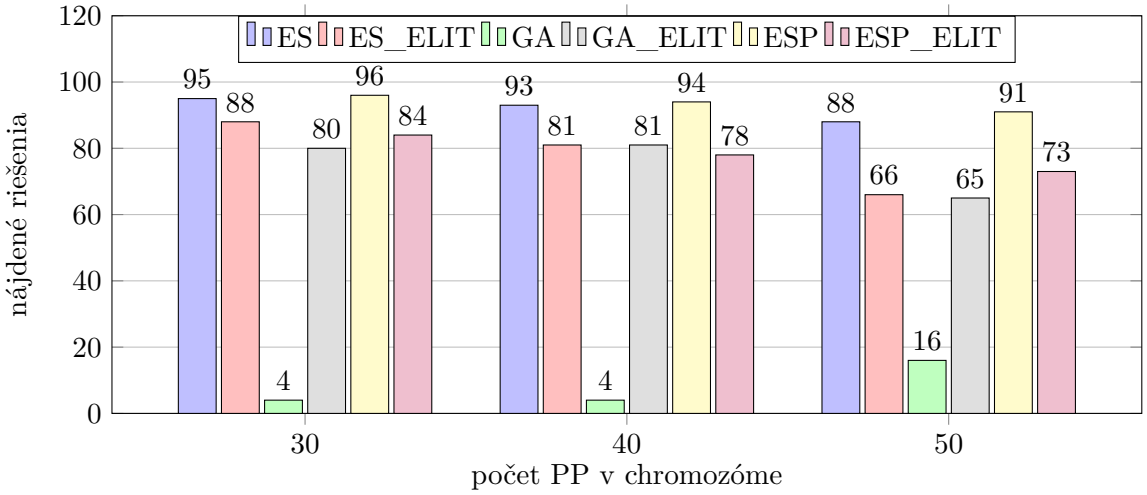
počet PP	úspešnosť	priemerná výp. náročnosť [kroky CA]	smerodajná odchýlka
30	85/96	412454.011	450132.27
40	92/96	337836.728	440756.86
50	91/96	364089.791	428342.75

Tabuľka 5.17: Výsledky experimentu s replikáciou vzoru

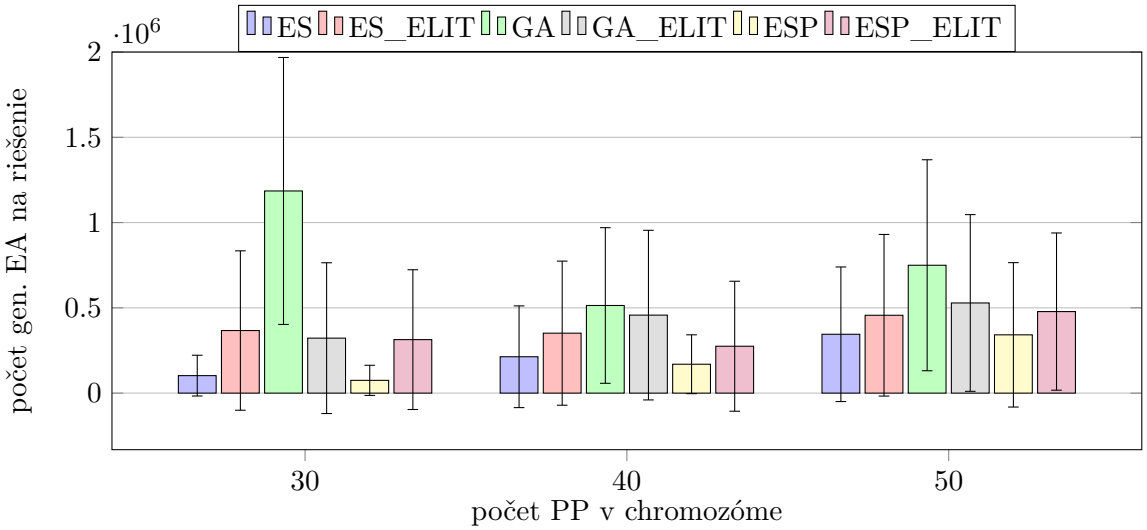
Pri použití ESP_ELIT na problém replikácie vzoru (tab. 5.17) sme dosiahli veľmi dobré výsledky, nielen čo sa týka úspešnosti, ale aj priemernej rýchlosti nájdenia riešení.

5.7 Diskusia

Experiment s vývojom vzoru, ktorého výsledky sú vidieť na grafoch 5.1 a 5.2, všetky algoritmy boli schopné nájsť riešenie. Poradie jednotlivých algoritmov, od najúspešnejšieho po najmenej úspešného, je: ESP, ES, ES_ELIT, ESP_ELIT, GA_ELIT a GA. Rozdiel medzi ES a ESP bol veľmi malý, len 5 riešení v prospech ESP, a dokonca mali veľmi podobnú výpočtovú náročnosť. Prekvapením boli veľmi zlé výsledky GA, ktorý v tomto experimente oproti ostatným algoritmom úplne prepadol.

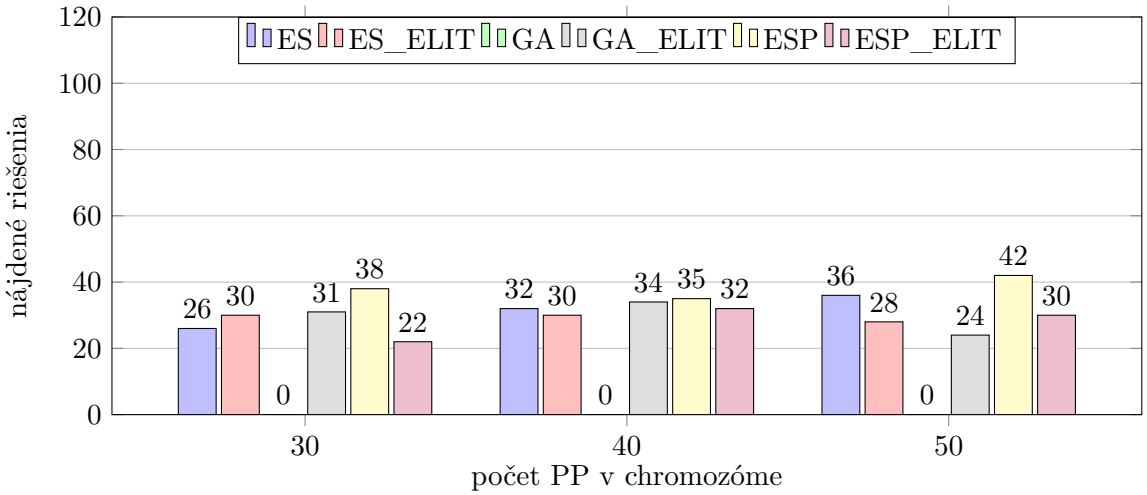


Obr. 5.1: Výsledky experimentu s vývojom vzoru

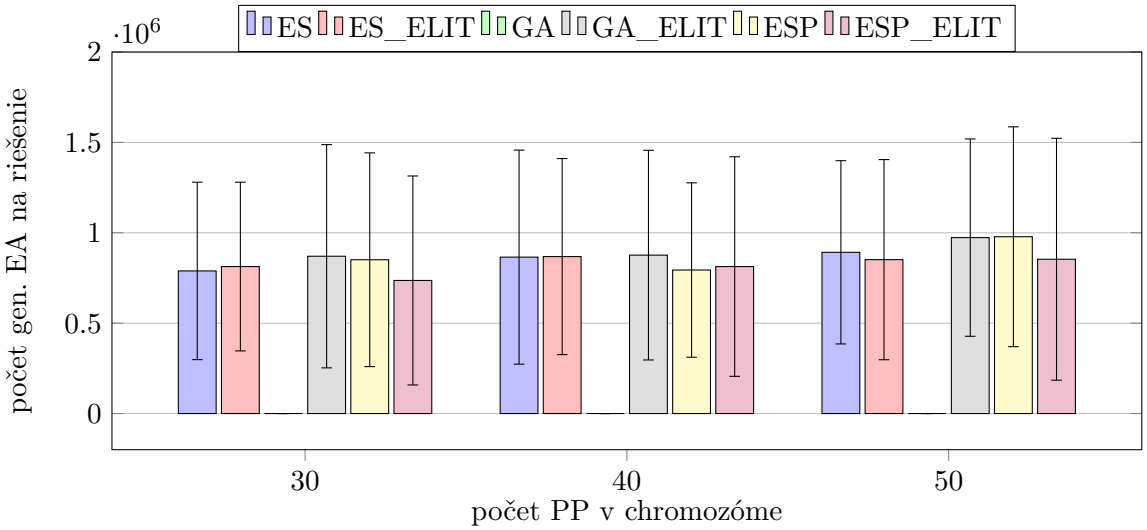


Obr. 5.2: Výpočtová náročnosť experimentu s vývojom vzoru.

Z úlohou nájdenia prechodovej funkcie, pri ktorej sa daný objekt správal ako glider, si poradili až na 1 všetky algoritmy. Algoritmus ktorému sa nepodarilo, ako je vidieť na grafe 5.3 a 5.4, nájsť riešenie je algoritmus GA. Podľa úspešnosti sa dajú algoritmy zoradiť takto: ESP, ES, GA_ELIT, ES_ELIT, ESP_ELIT a GA. Je dôležité spomenúť že rozdieli v úspešnosti a výpočtovej náročnosti boli veľmi malé. Z výsledkov je vidieť že táto úloha bola pre algoritmy náročná, lebo len tretina chodov EA úspešne našla riešenie.

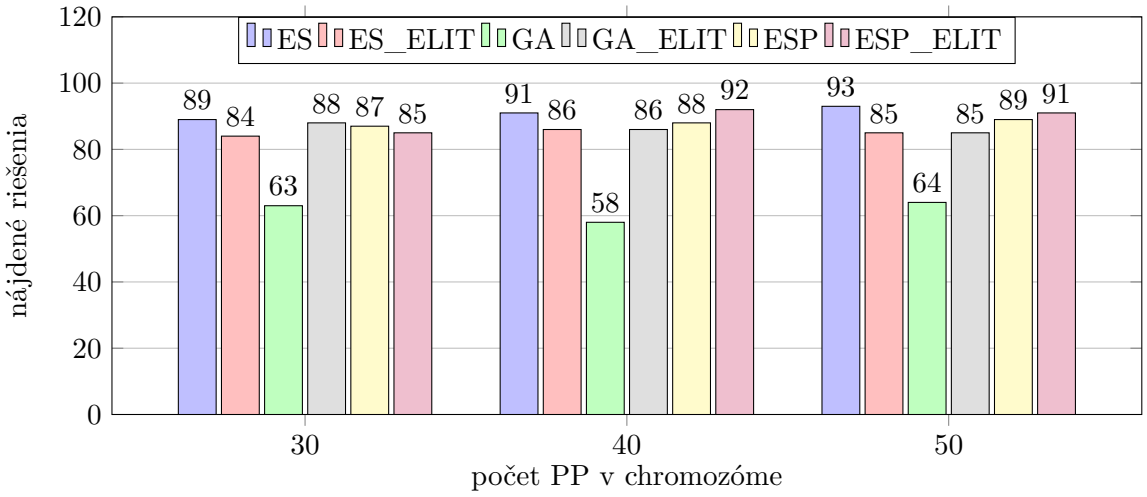


Obr. 5.3: Výsledky experimentu s gliderom

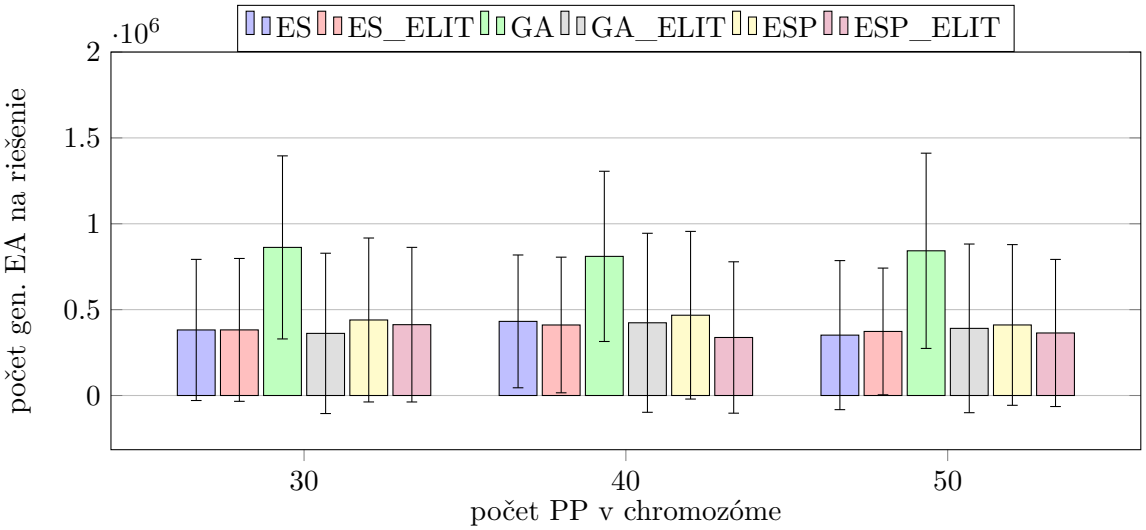


Obr. 5.4: Výpočtová náročnosť experimentu s gliderom.

Z grafov 5.5 a 5.6 je vidieť že všetky zvládli nájsť riešenia pre úlohu replikácie vzoru. Úspešnosť a výpočtová náročnosť jednotlivých algoritmov bola veľmi podobná, z výnimkou GA. GA našiel vždy viac ako o 20 riešení menej ako ostatné algoritmy a na ich nájdenie potreboval raz toľko času. Ak zoradíme jednotlivé algoritmy od najlepšieho, vyjde nám nasledujúce poradie: ES, ESP_ELIT, ESP, GA_ELIT, ES_ELIT a GA.



Obr. 5.5: Výsledky experimentu s replikáciou vzoru



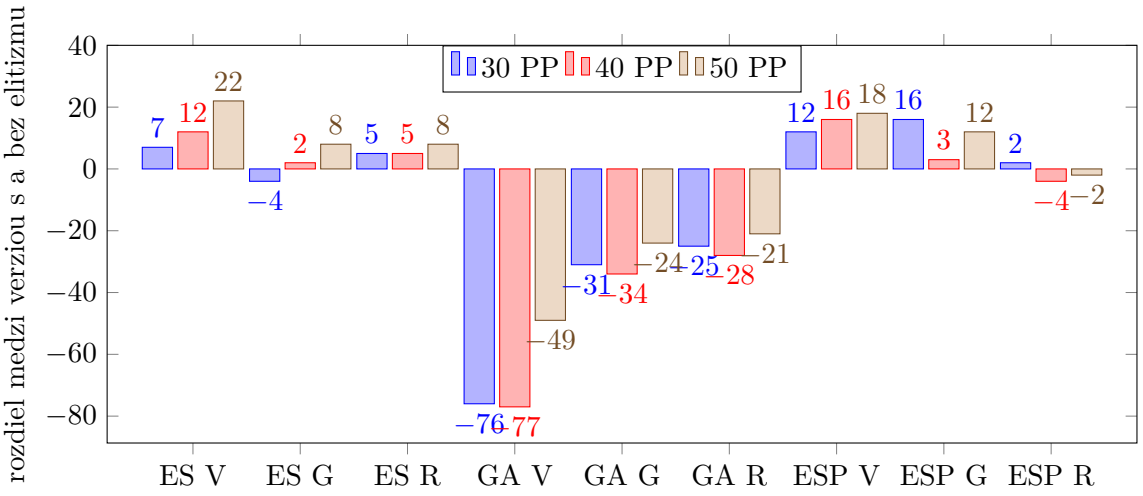
Obr. 5.6: Výpočtová náročnosť experimentu s replikáciou vzoru.

Ak sa pozrieme na výsledky zrovnaní jednotlivých experimentov, vyjde nám, že najlepšie si v jednotlivých úlohách viedol algoritmus ESP, tesne za ním ES a potom všetky ostatné. Rozdieli medzi ESP a ES, čo sa týka úspešnosti a výpočtovej náročnosti, neboli veľké, ale ESP nachádzalo bolo o trochu úspešnejšie. Ani rozdieli medzi verziami algoritmov s elitizmom neboli veľké, preto nie je možné z určitosťou povedať ktorý z nich je lepší.

Som veľmi prekvapený, že najhoršie dopadol GA bez elitizmu. GA vždy našiel najmenej riešení a potreboval najviac času na ich nájdenie. Dôvod môže byť niekoľko, ale ak sa zoberieme do úvahy úspešnosť verzie z elitizmom, tak to nasvedčuje buď na zlé nastavenie parametrov GA (napr. nastavenie parametrov turnaja) alebo na nevhodnosť GA pre daný priestor riešení.

Zaujalo ma, že ak sa pozrieme na grafy výpočtovej náročnosti jednotlivých experimentov (5.2, 5.4, 5.6), tak vidíme, že rozptyl v počte generácií bol obrovský. Toto bolo asi spôsobené náhodnou začiatočnou populáciou, ktorá zavinila to, že jednotlivé populácie nezačali rovnako ďaleko od riešení.

Vec ktorá ma zaujala je to, že najlepšie 2 algoritmy nepoužívajú elitizmus. Preto som sa rozhodol porovnať implementácie z elitizmom a bez neho. Na grafe 5.7 je vidieť, že ak bola implementácia postavená na evolučnej stratégií, tak mal elitizmus negatívne účinky. Naopak, ak sa jednalo o implementáciu GA, tak mal elitizmus veľmi pozitívne účinky.



Obr. 5.7: Porovnanie výhodnosti elitizmu. Kladná hodnota znamená, že verzia bez elitizmu bola úspešnejšia ako s elitizmom. Záporná hodnota znamená opak.

Kapitola 6

Záver

V tejto práci som naštudoval, navrhol a implementoval rôzne EA. Algoritmy ktoré som sa rozhodol implementovať boli verzie evolučnej stratégie a genetického algoritmu. Následne urobil porovnávaciu štúdiu týchto algoritmov na 3 experimentoch, a to experiment s vývojom vzoru, experiment s replikáciou vzoru a experiment z gliderom. Výsledky týchto experimentov som zhrnul v zhodnotil v kapitole 5.

Výsledky experimentov ukázali, že najúspešnejší algoritmus bol ESP, tesne za ním skončila ES, potom nasledovali verzie z elitizmom a to ES_ELIT, ESP_ELIT a GA_ELIT. Na poslednom mieste skončil GA, ktorému sa v jednom experimente nepodarilo nájsť ani jedno riešenie. Ďalej sa ukázalo, že na výsledky implementácií postavených na evolučnej stratégií (ES a ESP) pôsobí elitizmus negatívne. Pri GA však elitizmus značne zlepšil úspešnosť.

Ďalší výskum sa môže uberať niekoľkými smermi. Je tu možnosť pokračovať v téme tejto práce a urobiť podobnú porovnávaciu štúdiu na iných a ťažších problémoch. Ďalšiu možnosťou je použiť iné EA ako ES a GA na hľadanie prechodových funkcií. Alebo dokonca nepoužiť EA ale nejaký iný druh optimalizačných algoritmov, napr. mravčiu kolóniu.

Literatúra

[1] Ben-Menahem, A.: Historical Encyclopedia of Natural and Mathematical Sciences. ročník 1, 2009.

[2] Bentley, P.: *Evolutionary Design by Computers*. číslo pt. 1 in Evolutionary Design by Computers, Morgan Kaufman Publishers, 1999, ISBN 9781558606050.
URL `https://books.google.cz/books?id=EgC6LBAH5r8C`

[3] Bidlo, M.; Vasicek, Z.: Evolution of cellular automata with conditionally matching rules. In *2013 IEEE Congress on Evolutionary Computation*, June 2013, ISSN 1089-778X, s. 1178–1185, doi:10.1109/CEC.2013.6557699.

[4] Bidlo, M.; Vašíček, Z.: Evolution of cellular automata with conditionally matching rules. *2013 IEEE Congress on Evolutionary Computation*, 2013: s. 1178–1185, doi:10.1109/CEC.2013.6557699.

[5] Chowdhury, D. R.; Basu, S.; Gupta, I. S.; aj.: Design of CAECC - cellular automata based error correcting code. *IEEE Transactions on Computers*, ročník 43, č. 6, Jun 1994: s. 759–764, ISSN 0018-9340, doi:10.1109/12.286310.

[6] Durbeck, L. J. K.; Macias, N. J.: The Cell Matrix: an architecture for nanocomputing. *Nanotechnology*, ročník 12, č. 3, 2001: str. 217.
URL `http://stacks.iop.org/0957-4484/12/i=3/a=305`

[7] Gardner, M.: MATHEMATICAL GAMES - The fantastic combinations of John Conway's new solitaire game life. *Scientific American*, ročník 223, 1970: s. 120–123.

[8] Guillaume, A.; Lee, S.; Wang, Y. F.; aj.: Deep Space Network Scheduling Using Evolutionary Computational Methods. In *2007 IEEE Aerospace Conference*, March 2007, ISSN 1095-323X, s. 1–6, doi:10.1109/AERO.2007.352900.

[9] Hornby, G.; Globus, A.; Linden, D.; aj.: Automated Antenna Design with Evolutionary Algorithms. *Space 2006*, 2006, doi:10.2514/6.2006-7242.

[10] Jiří Pospíchal, P. T., Vladimír Kvasnička: *Evolučné algoritmy*. Bratislava : Slovenská technická univerzita v Bratislave vo Vydavateľstve STU, 2000, ISBN ISBN 80-227-1377-5.

[11] Lifewiki: Conway's Game of Life. [Online; cit. 2017-02-10.
URL `http://www.conwaylife.com/wiki/Conway's_Game_of_Life`

[12] von Neumann, J.: The general and logical theory of automata. *Cerebral mechanisms in behavior; the Hixon Symposium*, 1951: s. 1–41.

[13] Neumann, J. V.: *Theory of Self-Reproducing Automata*. Champaign, IL, USA: University of Illinois Press, 1966.

[14] Rajasekaran, S.: Cellular Automaton Model for Epidemiology with Vaccination Strategy. *International Journal of Infectious Diseases*, ročník 12, 2008, ISSN 1201-9712, doi:10.1016/j.ijid.2008.05.1265.

[15] Schiff, J. L.: *Cellular Automata - A Discrete View of the World*. Hoboken, N.J. : Wiley-Interscience, 2011, ISBN 9781118030639.

[16] Stoica, A.; Keymeulen, D.; Zebulum, R.; aj.: Evolution of analog circuits on field programmable transistor arrays. In *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*, 2000, s. 99–108, doi:10.1109/EH.2000.869347.

[17] Tomassini, M.; Sipper, M.; Perrenoud, M.: On the generation of high-quality random numbers by two-dimensional cellular automata. *IEEE Transactions on Computers*, ročník 49, č. 10, Oct 2000: s. 1146–1151, ISSN 0018-9340, doi:10.1109/12.888056.

[18] Whitley, D.: A genetic algorithm tutorial. *Statistics and Computing*, ročník 4, č. 2, 1994: s. 65–85, ISSN 1573-1375, doi:10.1007/BF00175354.
URL `http://dx.doi.org/10.1007/BF00175354`

[19] Wikipedia: Cellular automata hexagonal neighborhood. 2005, [Online; cit. 2017-02-10.
URL `https://en.wikibooks.org/wiki/File:Cellular_automata_hexagonal_neighborhood.png`

[20] Wolfram, S.: Statistical mechanics of cellular automata. *Reviews of Modern Physics*, ročník 55, č. 3, Jan 1983: str. 601–644, doi:10.1103/revmodphys.55.601.

Prílohy

Príloha A

Obsah priloženého pamäťového média

```
/
├── xorman00.pdf ... elektronická verzia tohto dokumentu
├── BP/ ... priečinok obsahujúci zdrojový tvar bakalárskej práce vo
│          formáte LaTeX
├── CA_evolution/ ... priečinok obsahujúci EA použité k návrhu prechodových
│   │                  funkcií CA
│   ├── src/ ... priečinok obsahujúci zdrojový kód programu CA_evolution
│   ├── tools/ ... priečinok obsahujúci pomocné skripty a nástroje
│   ├── sets/ ... priečinok obsahujúci nastavenia experimentov
│   ├── experiments/ ... priečinok obsahujúci vytvorené experimenty pripravené na
│   │                    spustenie
│   └── Makefile ... makefile umožnujúci základné správu experimentov
└── CA_visualiser/ ... priečinok obsahujúci nástroj na zobrazovanie CA, ktorý bol
    │                  použitý na zobrazenie a kontrolu nájdených prechodových
    │                  funkcií
    ├── OrCAS ... nástroj umožnujúci vyzualizáciu CA
    └── config.cfg ... konfigurácia nástroja OrCAS
```

Ako je vidieť z obsahu, táto práca obsahuje 2 aplikácie implementované v jazykoch C a Python 3 v prostredí operačného systému Linux. Natívnym prostredím príkladov spustenia je príkazový riadok (tzv. shell).

Príloha B

Program CA_evolution

Tento program slúži k evolučnému návrhu prechodových funkcií CA a je základným kameňom tejto práce. Program využíva možnosti štandardnej knižnice jazyka C, ale pomocné skripty pre svoju správnu činnosť potrebujú Python 2 a knižnicu numpy.

Program je primárne určený na spustenie na superpočítače ako ANSELM alebo SALOMON. Pri takomto použití sa program ovláda pomocou Makefile:

- `make` alebo `make bp` - Vytvorí experimenty použité v tejto porovnávacej štúdii.
- `make exp` - Vytvorí experimentálne sady použité pri vývoji.
- `make run` - Vybuilduje experimenty v priečinku *experiments/* a spustí ich.
- `make stop` - Predčasne zastaví spustené experiment.
- `make results` - Spravuje výsledky experimentov, nájdené prechodové uloží do súborov vo formáte json do priečinku *_results/* a vypíše štatistiky.
- `make delete` - Vymaže experimenty v priečinku *experiments/* a priečinok *_results/*.

Program je však možné spustiť aj priamo, a to pomocou príkazu `make` v priečinku *src/*, ktorý vytvorí spustiteľný binárny súbor. Program má 2 vstupné argumenty, prvý argument je cesta k súboru zo začiatočným stavom CA a druhý argument je cesta k súboru s cieľovým stavom CA. Program môže byť spustený napríklad takto:

```
$ evol beg.txt target.txt > results.txt
```

Uvedený príkaz zároveň zaistí presmerovanie výsledkov do súboru z názvom *results.txt*, namiesto vypísanie výsledkov na štandardný výstup.

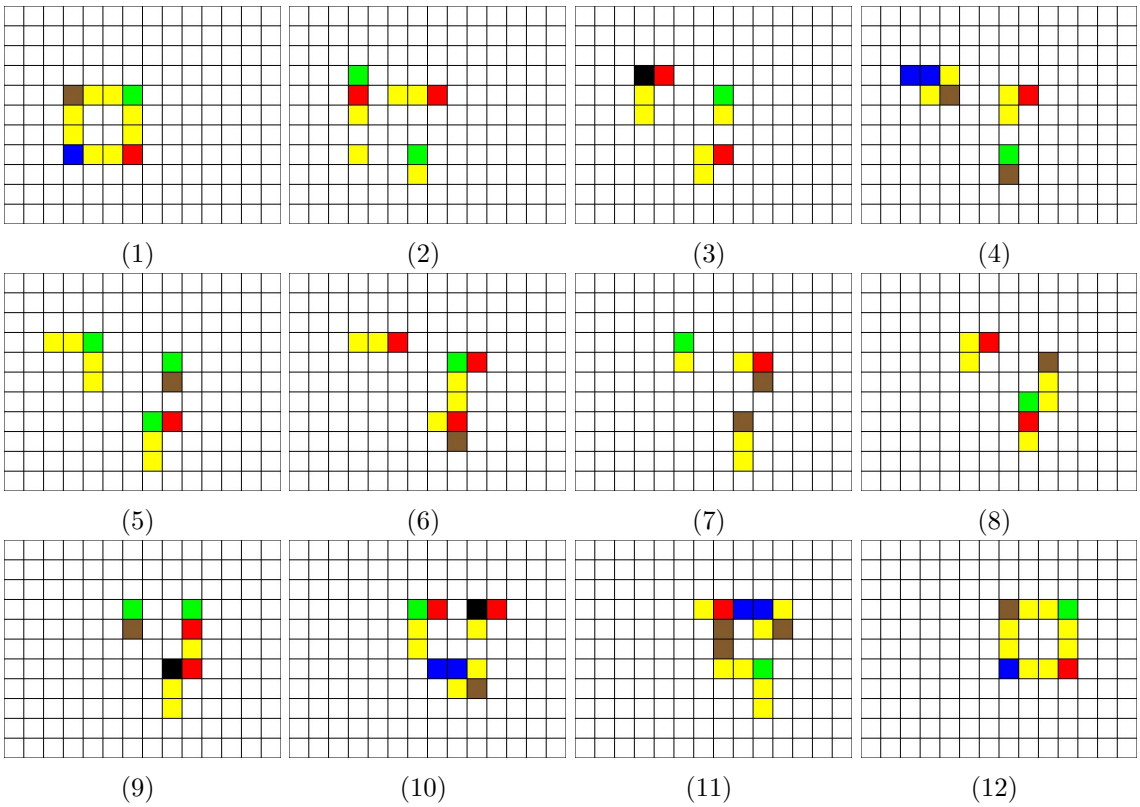
Program sa nastavuje pomocou makier v súboroch *src/params.h* a *src/local_params.h*, ktoré sú nasledovné:

- `WIDTH_PARAM` - Šírka CA.
- `HEIGHT_PARAM` - Výška CA.
- `CMR_COUNT` - Počet PP v chromozóme.
- `POPULATION_SIZE` - Veľkosť populácie CA.
- `STATES_COUNT` - Počet stavov CA.

- `NON_EVAL_CYCLES` - Počet nevyhodnocovaných krokov CA.
- `CYCLES` - Celkový počet krokov CA.
- `GENERATIONS` - Počet generácií EA.
- `MAX_MUTATIONS` - Maximálny počet mutácií na potomka.
- `MIN_MUTATIONS` - Minimálny počet mutácií na potomka.
- `TURNAMENT_ROUNDS` - Počet turnajových kôl.

To ktorý EA bude použitý pre evolúciu je ovládané nasledujúcimi makrami:

- `ES` - Použi algoritmus ES.
- `ES_ELIT` - Použi ES s elitizmom.
- `GA` - Použi algoritmus GA.
- `GA_ELIT` - Použi GA s elitizmom.
- `ESP` - Použi algoritmus ESP.
- `ESP_ELIT` - Použi ESP s elitizmom.



Obr. B.1: Glider nájdený pomocou CA_evolution.

Príloha C

Nástroj OrCAS

Tento nástroj je možné použiť na vizualizáciu riešení ktoré boli nájdene pomocou programu CA_evolution. Nástroj je napísaný v jazyku Python 3 a na jeho spustenie je nutná externá knižnica pygame. Program je možné spustiť napr. takto:

```
$ OrCAS -config config.cfg -json result.json
```

Nástroj pozná nasledujúce argument:

- *-r, --rows* - Počet riadkov CA.
- *-c, --cols* - Počet stĺpcov CA.
- *--config* - Cesta ku konfiguračnému súboru. Je vhodné použiť priložený súbor config.cfg.
- *--json* - Cesty k súborom s prechodovými funkciami, napr. obsah priečinku *_result*.

Po spustení sa program ovláda nasledovnými klávesami:

- *hore* - Ďalšie prechodová funkcia.
- *dole* - Predchádzajúca prechodová funkcia.
- *vpravo* - Ďalší krok CA.
- *vľavo* - Predchádzajúci krok CA.
- *c* - Resetne CA.
- *p* - Vytvorí obrázok so momentálnym stavom CA. Obrázok bude mať meno <krok CA>.jpeg
- *escape* - Ukončí program.